Inside Convolutional Networks

AW

Lecture Overview

1. **Convolution Operation**

1. **Backpropagation in CNN**

The Convolution Operation

- Let $x(t)$ tracking plane position;
  - tracking is noisy so we want to average over time and give more weight to most recent observations. Use weights $w(a)$, so contribution of observation at a time a is $x(a)w(t-a)$ where t is current time
  - If we average over time this is

$$s(t) = \int_{-\infty}^{\infty} x(a)w(t-a)da$$

- It is convolution denoted $s(t) = x * w(t)$. Here x is said to be *input* and w is said to be *kernel*
- In this example w needs to be a valid probability density function- otherwise $s(t)$ is not a weighted average

The Convolution Operation – Discrete Variant

- Often time is discrete e.g. day 1, day 2,…, or frame 1,frame 2,…. Then the convolution operation is discrete:

$$s(t) = \sum_{a=-\infty}^{\infty} x(a) \times w(t-a) = x * w(t)$$

- In neural networks data is normally tensor and weights (parameters) are normally also tensor of the same dimension (or one less).
- When we are averaging we are averaging either within a batch (or over a spatial structure)
- One of the dimensions of a tensor is represents time (or the number of an example within a batch, or a position within filter), so each time slice of the input and kernel must be explicitly stored separately. Thus the only way to understand time is that these functions are zero everywhere but the a finite set of points for which we stored the tensor values.

The Convolution Operation – Discrete Variant

- Convolutions often are used over more than one axis at a time.
  - Example: Input is two-dimensional image I. Then kernel K must also be two-dimensional. Then we need 2-dim convolution:

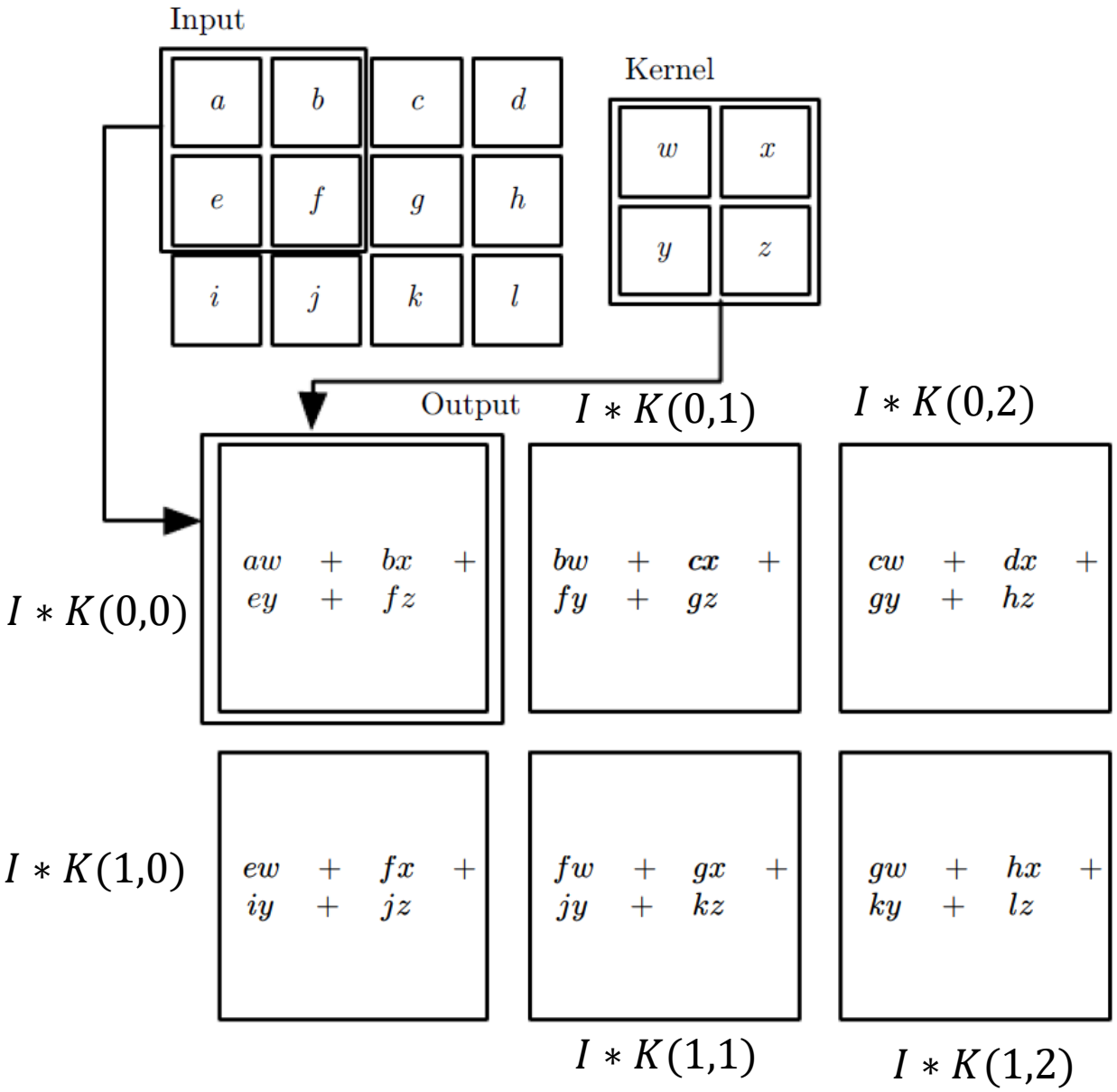
$$S(t,r) = I * K(t,r) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} I(i,j) \times K(t-i, r-j)$$
$$= \sum_{i}^{\infty} \sum_{j}^{\infty} I(t-i, r-j) \times K(i,j)$$
$$= K * I(t,r)$$

so convolution is commutative.

  - It is also sometimes its is called cross-correlation, when indexed increasing into I which is clearly the same:

$$S(t,r) = I * K(t,r) = \sum_{i}^{\infty} \sum_{j}^{\infty} I(t+i, r+j) * K(i,j)$$

Example of 2-D Convolution Application (into I)



Real-world Convolution Operation

- When working with images, the input and output of the convolution as being 3D tensors, with one index into the different channels and two indices into the spatial coordinates of each channel
- Forward track and backward propagation work with weights that are multiply output (color or filter type) channel k previous layer i^{th} neuron to become input to channel l of the next layer j^{th} neuron thus making it 6D tensor
- Plus data may come in batches which makes it 4D and 7D tensors respectively
- Note that tensors are not of the same dimensions

Convolution for Real-world Tensors

We describe convolution of input V that is 3D tensor (picture output of previous layer) with filter (kernel) K that is 4D tensor into the output of (next) convolutional layer where

- element v_{ipq} of V is a channel i output element (neuron that outputs) in row p column q
- Element k_{ijmn} of K is the weight (strength of input) from channel i to channel j with an offset of m rows and n columns between the output unit and the input unit

- The result of the convolution is 3D tensor $Z = V * K$ where z_{ijk} entry of the tensor is the output of the channel j at row s, column t. The convolution is given by

$$z_{jst} = \sum_i \sum_m \sum_n v_{i,s-1+m,t-1+n} k_{i,j,m,n}$$

  where $p = s - 1 + m$ and $q = t - 1 + m$
- In linear algebraic notation, tensors are indexes starting from 1 which explains −1.
- In fact it is 2 dimensional convolution with the summation over the 3rd axis after convolution and it is done for every value of 4th axis

Convolution for Real-world Tensors (cont)

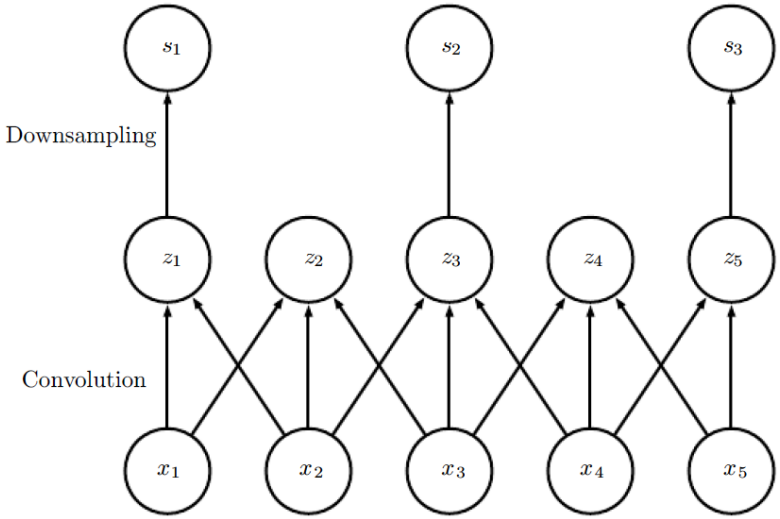
- Often implemented through 2D convolutions
- On 2D tensors computation is Hadamard product. Denoted by $\odot$:
  - Flatten matrix into vectors
  - Perform dot product of flattened vectors
  - Reshape back into matrix, e.g.

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \odot \begin{pmatrix} e & f \\ g & h \end{pmatrix} \Rightarrow \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} \cdot \begin{pmatrix} e \\ f \\ g \\ h \end{pmatrix} = ae + bf + cg + dh$$

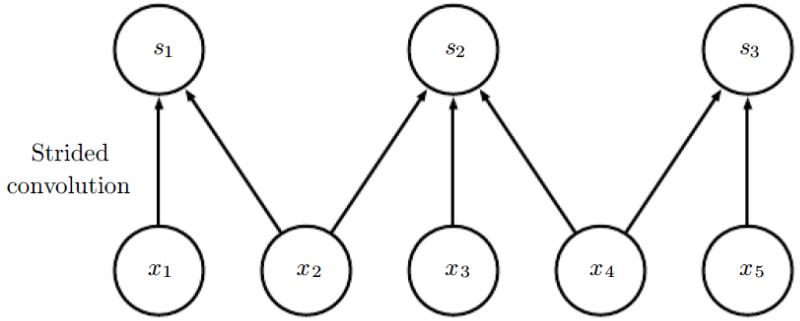
- In real world more often performed as multiplication of precomputed matrix shapes (2D tensors) - later

Stride Formally

- Skip over some positions of the kernel in order to reduce the computational cost (at the expense of not extracting our features) which is down-sampling the output of the full convolution function.



Convolution with down-sampling



Implemented as single operation

Valid , Strided , Local Variants of Convolution

- If we to sample only every u pixels in each direction in the output, then u is the stride and formal *strided convolution* $Z = c(V * K, u)$ is

$$z_{ist} = \sum_i \sum_m \sum_n v_{i,m+(s-1)u,n+(t-1)u} k_{i,j,m,n}$$

- It is also possible to define different stride u_d in each direction d
- Ordinary convolution without padding is also know as *valid convolution,* with padding it is called *full convolution*
- Notice that in valid, full and in strided convolution filter applied to all inputs of one layer of one channel is the same
- The size of the filter is much smaller than the size of the layer

Locally Connected Convolution

In *local convolution*:

- To each window its own filter is applied. In other words the adjacency matrix of the CNN graph is the not the same as either in valid or in full convolution
- A channel local convolution (CLC) kernel is characterized by its channel dependency graph (CDG), which is an acyclic graph where the nodes represent channels and edges represent dependencies
- To each window $k \times k \times d$ filter is applied which essentially means that there are d different filters, each of which will (learn to) have different weights.
  - Each filter will detect the presence of a particular pattern across a feature map (e.g. an input image) but different filters will likely learn to detect unrelated features.

Locally Connected Convolution

In *local convolution*: Since every connection has its own weight, specified by a 6D tensor W with elements $w_{j,s,t,i,m,n}$ where

- j indexes the output channel,
- s indexes the output row for a filter within a size l group,
- t indexes the output column for a filter within a size l group,
- i indexes the input channel,
- m indexes the row offset within the input,
- n indexes the column offset within the input so

$$z_{ist} = \sum_i \sum_m \sum_n v_{i,m+s-1,n+t-1} w_{j,s\%l+1,t\%l+1,i,m,n}$$

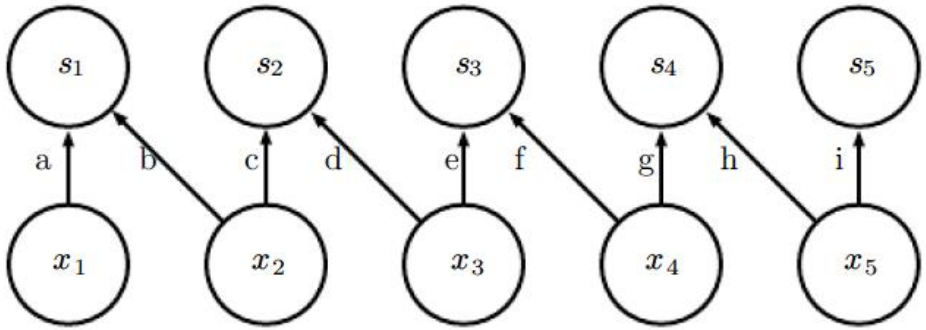
Tiled Convolution

- *Tiled convolution* learns a set of kernels that rotate through as we move through the input space.
- Immediately neighboring locations have different filters, like in a locally connected layer, but the memory requirements for storing the parameters increase only by a factor of the size of this set of kernels, rather than the size of the entire output feature map.
- In this case weights are given by 6D tensor W where element $w_{i,j,m,l,r}$ is as in local

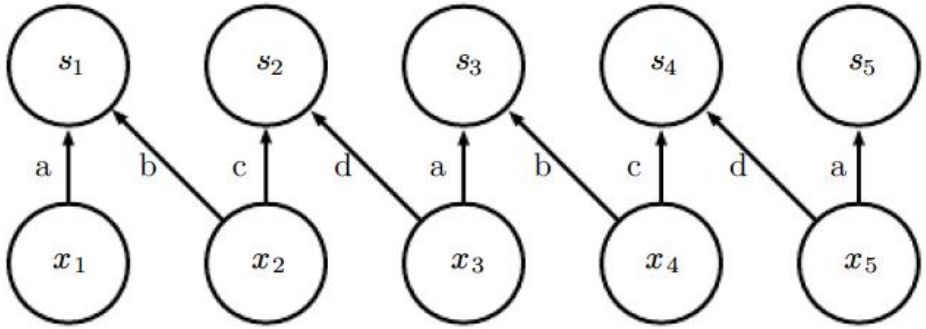
$$z_{ijt} = \sum_{l} \sum_{m} \sum_{n} v_{l,m+j-1,n+k-1} w_{i,j,k,l,m,n}$$

Difference between types of Convolution

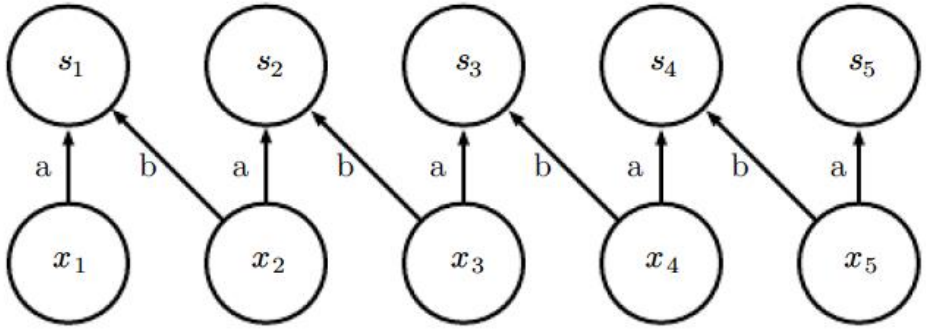
Local convolution with window size 2



Tiled convolution with window size 2



Full convolution with window size 2



Lecture Overview

1. Convolution Operation

2. Backpropagation in CNN

Bird's Eye View: Backpropagating Convolutions

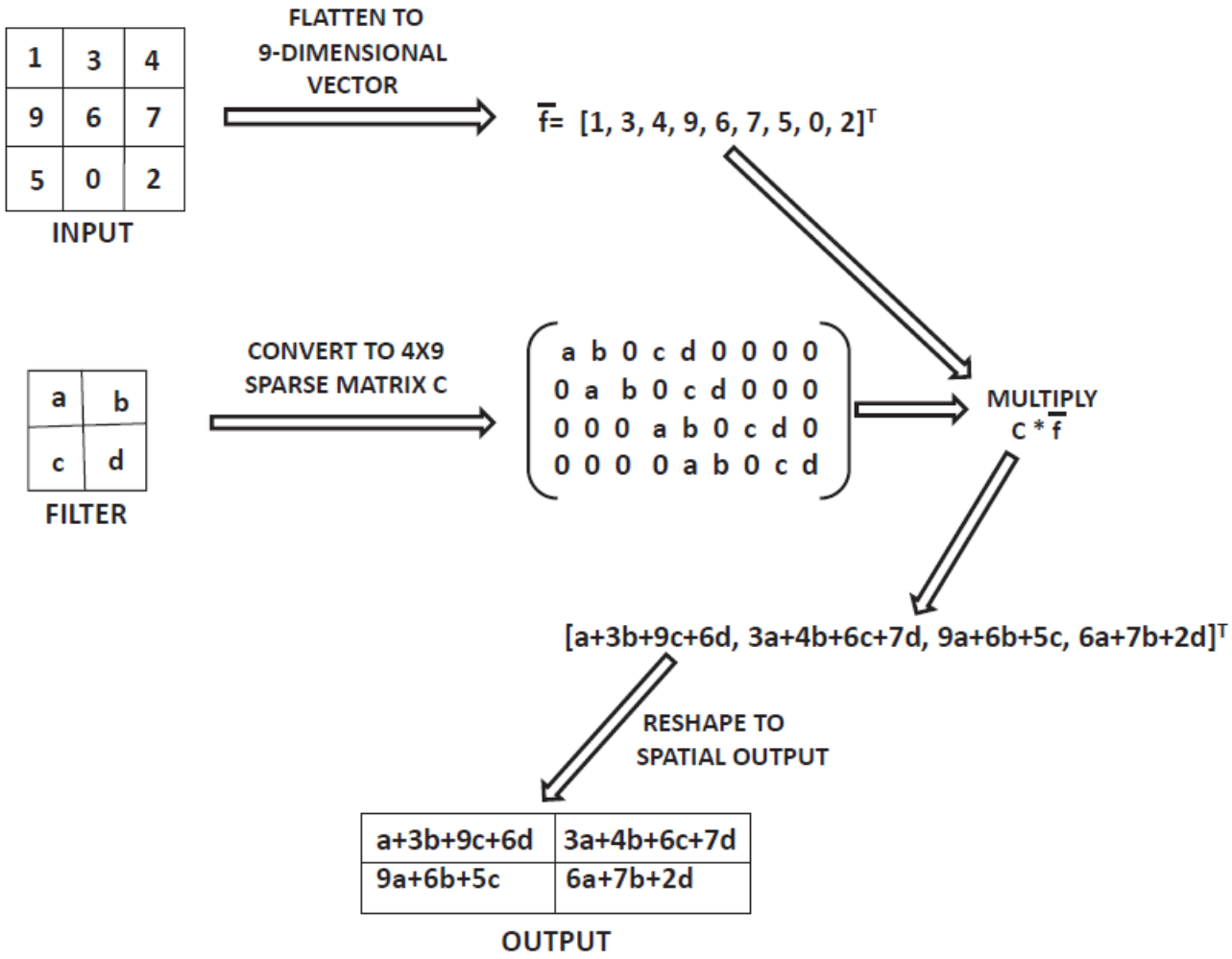
- Traditional backpropagation is transposed matrix multiplication.
- Backpropagation through convolutions is transposed convolution (i.e., with an *inverted filter*).
  - It can and often is implemented as matrix multiplication
- Derivative of loss with respect to each cell is backpropagated.
  - Elementwise approach of computing which cell in input contributes to which cell in output.
  - Multiplying with an inverted filter.
- Convert layer-wise derivative to weight-wise derivative and add over shared weights.

Technicalities of ⊙ Implementation

- Convolution can be presented as a matrix multiplication
  - Useful during forward and backward propagation.
  - Backward propagation can be presented as transposed matrix multiplication
- Precomputed matrix shapes for filter sizes and image sizes:
  - Tensors of defined shapes that are computed with the standard tabulation

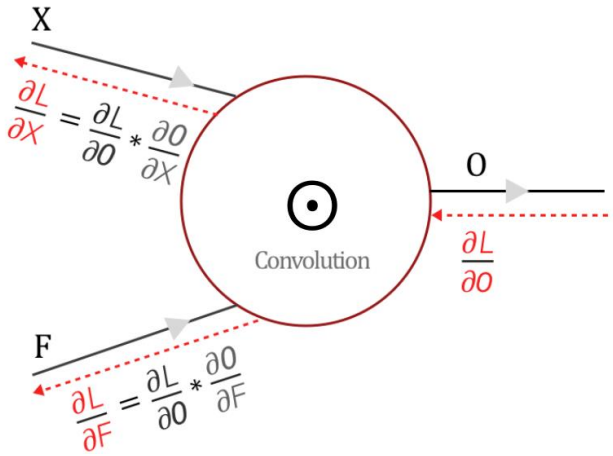
Example of Converting * to ×

Example: Filter size 3×3, picture block size 9×9



Backpropagation by Example - Settings

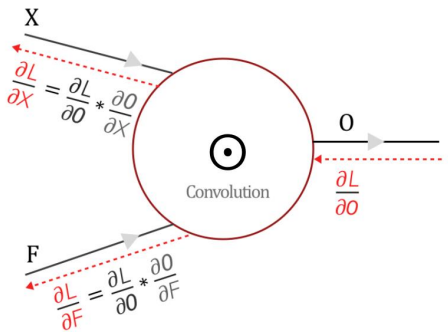
- At the end of forward pass loss L was obtained
- It is worked backwards, layer across layer, so the gradient of the loss from the previous layer $\frac{\partial L}{\partial o}$ is returned to our convolution neuron. To be propagated to the other gates, we need to find $\partial L/\partial X$ and we need $\partial L/\partial F$ for filter updates.



- Suppose Convolution neuron has input image X be 3×3 and uses filter F be 2×2 and we assume no padding at all

Backpropagation by Example – Forward Result

$$X = \begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{pmatrix}, F = \begin{pmatrix} F_{11} & F_{12} \\ F_{21} & F_{22} \end{pmatrix}$$



Forward we obtain output $O = \begin{pmatrix} o_{11} & o_{12} \\ o_{21} & o_{22} \end{pmatrix}$

where
$$o_{11} = F_{11}x_{11} + F_{12}x_{12} + F_{21}x_{21} + F_{22}x_{22}$$
$$o_{12} = F_{11}x_{12} + F_{12}x_{13} + F_{21}x_{22} + F_{22}x_{23}$$
$$o_{21} = F_{11}x_{21} + F_{12}x_{22} + F_{21}x_{31} + F_{22}x_{32}$$
$$o_{22} = F_{11}x_{22} + F_{12}x_{23} + F_{21}x_{32} + F_{22}x_{33}$$

Just like before for weights we need $\frac{\partial L}{\partial F}$ to update the filter. By chain rule

$$\frac{\partial L}{\partial F} = \begin{pmatrix} \frac{\partial L}{\partial F_{11}} & \frac{\partial L}{\partial F_{12}} \\ \frac{\partial L}{\partial F_{21}} & \frac{\partial L}{\partial F_{22}} \end{pmatrix}$$ where by chain rule we have

$$\frac{\partial L}{\partial F_{ij}} = \frac{\partial L}{\partial o} \times \frac{\partial o}{\partial F_{ij}} = \underbrace{\begin{pmatrix} \frac{\partial L}{\partial o_{11}} & \frac{\partial L}{\partial o_{12}} \\ \frac{\partial L}{\partial o_{21}} & \frac{\partial L}{\partial o_{22}} \end{pmatrix}}_{\text{Gradient from previous layer}} \odot \begin{pmatrix} \frac{\partial o_{11}}{\partial F_{ij}} & \frac{\partial o_{12}}{\partial F_{ij}} \\ \frac{\partial o_{21}}{\partial F_{ij}} & \frac{\partial o}{\partial F_{ij}} \end{pmatrix}$$

Example: Computation of Filter Gradient

Applying $\frac{\partial L}{\partial F_{ij}} = \begin{pmatrix} \frac{\partial L}{\partial o_{11}} & \frac{\partial L}{\partial o_{12}} \\ \frac{\partial L}{\partial o_{21}} & \frac{\partial L}{\partial o_{22}} \end{pmatrix} \odot \begin{pmatrix} \frac{\partial o_{11}}{\partial F_{ij}} & \frac{\partial o_{12}}{\partial F_{ij}} \\ \frac{\partial o_{21}}{\partial F_{ij}} & \frac{\partial o}{\partial F_{ij}} \end{pmatrix}$

we get $\frac{\partial L}{\partial F_{11}} = \frac{\partial L}{\partial o_{11}} x_{11} + \frac{\partial L}{\partial o_{12}} x_{12} + \frac{\partial L}{\partial o_{21}} x_{21} + \frac{\partial L}{\partial o_{22}} x_{22}$

$\frac{\partial L}{\partial F_{12}} = \frac{\partial L}{\partial o_{11}} x_{12} + \frac{\partial L}{\partial o_{12}} x_{13} + \frac{\partial L}{\partial o_{21}} x_{22} + \frac{\partial L}{\partial o_{22}} x_{23}$

$\frac{\partial L}{\partial F_{21}} = \frac{\partial L}{\partial o_{11}} x_{21} + \frac{\partial L}{\partial o_{12}} x_{22} + \frac{\partial L}{\partial o_{21}} x_{31} + \frac{\partial L}{\partial o_{22}} x_{32}$

$\frac{\partial L}{\partial F_{22}} = \frac{\partial L}{\partial o_{11}} x_{22} + \frac{\partial L}{\partial o_{12}} x_{23} + \frac{\partial L}{\partial o_{21}} x_{32} + \frac{\partial L}{\partial o_{22}} x_{33}$

So we get convolution $\begin{pmatrix} \frac{\partial L}{\partial F_{11}} & \frac{\partial L}{\partial F_{12}} \\ \frac{\partial L}{\partial F_{21}} & \frac{\partial L}{\partial F_{22}} \end{pmatrix} = \underbrace{\underbrace{\begin{pmatrix} \frac{\partial L}{\partial o_{11}} & \frac{\partial L}{\partial o_{12}} \\ \frac{\partial L}{\partial o_{21}} & \frac{\partial L}{\partial o_{22}} \end{pmatrix}}_{\text{Gradient from previous layer}} * \begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{pmatrix}}_{convolution}$

In other words $\frac{\partial L}{\partial F} = \frac{\partial L}{\partial o} * X$

Example: Computation of Propagated Gradient

$$\frac{\partial L}{\partial X} = \begin{pmatrix} \frac{\partial L}{\partial x_{11}} & \frac{\partial L}{\partial x_{12}} & \frac{\partial L}{\partial x_{13}} \\ \frac{\partial L}{\partial x_{21}} & \frac{\partial L}{\partial x_{22}} & \frac{\partial L}{\partial x_{23}} \\ \frac{\partial L}{\partial x_{31}} & \frac{\partial L}{\partial x_{32}} & \frac{\partial L}{\partial x_{33}} \end{pmatrix} \text{ where}$$

$$\frac{\partial L}{\partial x_{ij}} = \begin{pmatrix} \frac{\partial L}{\partial o_{11}} & \frac{\partial L}{\partial o_{12}} \\ \frac{\partial L}{\partial o_{21}} & \frac{\partial L}{\partial o_{22}} \end{pmatrix} \odot \begin{pmatrix} \frac{\partial o_{11}}{\partial x_{ij}} & \frac{\partial o_{12}}{\partial x_{ij}} \\ \frac{\partial o_{21}}{\partial x_{ij}} & \frac{\partial o_{22}}{\partial x_{ij}} \end{pmatrix}. \text{ Computing output gradients yields}$$

$$\frac{\partial L}{\partial x_{11}} = \frac{\partial L}{\partial o_{11}} F_{11}, \frac{\partial L}{\partial x_{12}} = \frac{\partial L}{\partial o_{11}} F_{12} + \frac{\partial L}{\partial o_{11}} F_{11}, \frac{\partial L}{\partial x_{13}} = \frac{\partial L}{\partial o_{12}} F_{12}$$

$$\frac{\partial L}{\partial x_{21}} = \frac{\partial L}{\partial o_{11}} F_{21} + \frac{\partial L}{\partial o_{21}} F_{11}, \frac{\partial L}{\partial x_{22}} = \frac{\partial L}{\partial o_{21}} F_{12} + \frac{\partial L}{\partial o_{22}} F_{11}, \frac{\partial L}{\partial x_{23}} = \frac{\partial L}{\partial o_{12}} F_{22} + \frac{\partial L}{\partial o_{22}} F_{12}$$

$$\frac{\partial L}{\partial x_{31}} = \frac{\partial L}{\partial o_{21}} F_{21}, \frac{\partial L}{\partial x_{32}} = \frac{\partial L}{\partial o_{21}} F_{22} + \frac{\partial L}{\partial o_{22}} F_{21}, \frac{\partial L}{\partial x_{33}} = \frac{\partial L}{\partial o_{22}} F_{22}$$

Example: Propagated Gradient continued

1. The $\frac{\partial L}{\partial X}$ is a matrix of the same dimension as the image (3×3 matrix in our case) so if as before we want it to be a convolution of $\frac{\partial L}{\partial o}$ matrix with some filter then the input matrix has to be padded by $\left\lfloor \frac{F_q}{2} \right\rfloor$ picture-frame of 0's, so add a row of 0's above and below and a column of 0's before and after. Denote this matrix $\left(\frac{\partial L}{\partial o}\right)_p$
2. If there is convolution-based implementation of equations below what is the filter that produced it?

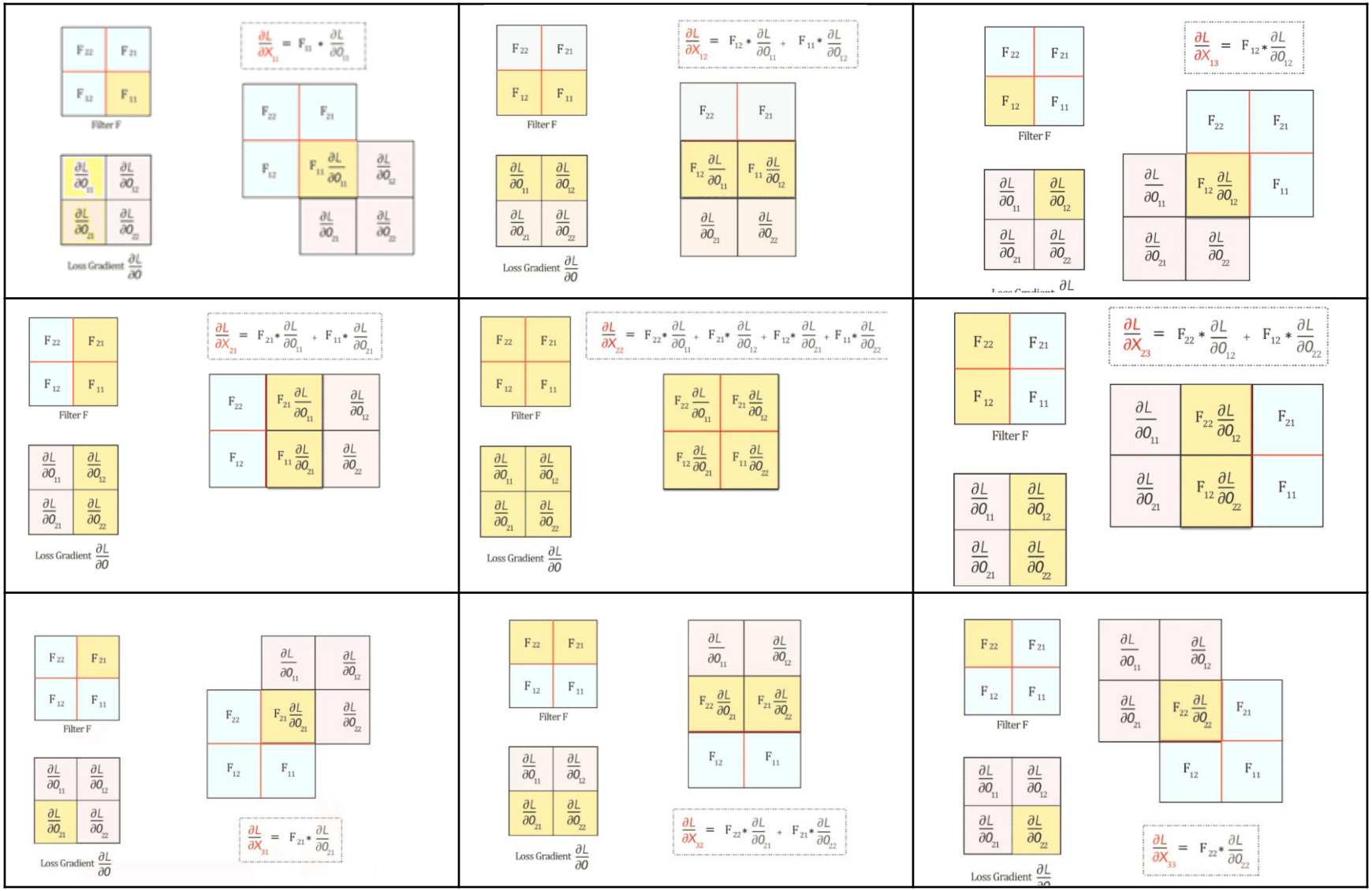
$$\frac{\partial L}{\partial x_{11}} = \frac{\partial L}{\partial o_{11}} F_{11}, \frac{\partial L}{\partial x_{12}} = \frac{\partial L}{\partial o_{11}} F_{12} + \frac{\partial L}{\partial o_{11}} F_{11}, \frac{\partial L}{\partial x_{13}} = \frac{\partial L}{\partial o_{12}} F_{12}$$

$$\frac{\partial L}{\partial x_{21}} = \frac{\partial L}{\partial o_{11}} F_{21} + \frac{\partial L}{\partial o_{21}} F_{11}, \frac{\partial L}{\partial x_{22}} = \frac{\partial L}{\partial o_{21}} F_{12} + \frac{\partial L}{\partial o_{22}} F_{11},$$

$$\frac{\partial L}{\partial x_{23}} = \frac{\partial L}{\partial o_{12}} F_{22} + \frac{\partial L}{\partial o_{22}} F_{12}$$

$$\frac{\partial L}{\partial x_{31}} = \frac{\partial L}{\partial o_{21}} F_{21}, \frac{\partial L}{\partial x_{32}} = \frac{\partial L}{\partial o_{21}} F_{22} + \frac{\partial L}{\partial o_{22}} F_{21}, \frac{\partial L}{\partial x_{33}} = \frac{\partial L}{\partial o_{22}} F_{22}$$

Reading


Filter F
Loss Gradient $\frac{\partial L}{\partial O}$
$\frac{\partial L}{\partial X_{11}} = F_{11} * \frac{\partial L}{\partial O_{11}}$
$\frac{\partial L}{\partial X_{12}} = F_{12} * \frac{\partial L}{\partial O_{11}} + F_{11} * \frac{\partial L}{\partial O_{12}}$
$\frac{\partial L}{\partial X_{13}} = F_{12} * \frac{\partial L}{\partial O_{12}}$
$\frac{\partial L}{\partial X_{21}} = F_{21} * \frac{\partial L}{\partial O_{11}} + F_{11} * \frac{\partial L}{\partial O_{21}}$
$\frac{\partial L}{\partial X_{22}} = F_{22} * \frac{\partial L}{\partial O_{11}} + F_{21} * \frac{\partial L}{\partial O_{12}} + F_{12} * \frac{\partial L}{\partial O_{21}} + F_{11} * \frac{\partial L}{\partial O_{22}}$
$\frac{\partial L}{\partial X_{23}} = F_{22} * \frac{\partial L}{\partial O_{12}} + F_{12} * \frac{\partial L}{\partial O_{22}}$
$\frac{\partial L}{\partial X_{31}} = F_{21} * \frac{\partial L}{\partial O_{21}}$
$\frac{\partial L}{\partial X_{32}} = F_{22} * \frac{\partial L}{\partial O_{21}} + F_{21} * \frac{\partial L}{\partial O_{22}}$
$\frac{\partial L}{\partial X_{33}} = F_{22} * \frac{\partial L}{\partial O_{22}}$

Downstream by Convolution with Inverted Filter

- So we have seen the filter that worked: $F_{inv} = \begin{pmatrix} F_{22} & F_{21} \\ F_{12} & F_{11} \end{pmatrix}$
- This is what is called *inverted filter* (not inverse!) It is obtained from original filter by placing the center in the middle cell of square matrix (or if it has even dimension right in the middle of the matrix) and flipping it first horizontally and then vertically

Example:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \rightarrow \begin{pmatrix} a_{13} & a_{12} & a_{11} \\ a_{23} & a_{22} & a_{21} \\ a_{33} & a_{32} & a_{31} \end{pmatrix} \rightarrow \begin{pmatrix} a_{33} & a_{32} & a_{31} \\ a_{23} & a_{22} & a_{21} \\ a_{13} & a_{12} & a_{11} \end{pmatrix}$$

- Then $\frac{\partial L}{\partial X} = \underbrace{\underbrace{\left(\frac{\partial L}{\partial o}\right)_p}_{\text{Padded gradient of previous layer}} * F_{inv}}_{convolution}$

Reading

- Ch. 8.3